

Algoritmos e Estruturas de Dados II

Trabalho Prático 2

Entrega: 01/10/09

Devolução: 22/10/08

Trabalho individual

Prof. Jussara Marques de Almeida

Simulação é uma técnica muito utilizada para avaliação de desempenho de sistemas de computação. Um simulador cria uma abstração do sistema alvo, capturando seus aspectos mais relevantes para o estudo em questão. Neste trabalho prático, o seu objetivo é estudar o desempenho relativo de diferentes *políticas de escalonamento* visando o gerenciamento de recursos em um sistema distribuído chamado NeoLook.

Sistema alvo:

O sistema alvo de estudo é o sistema NeoLook da empresa HG, uma plataforma distribuída para armazenamento e processamento de dados. O NeoLook é composto de N computadores interligados por uma rede de alta velocidade. Cada computador contém uma CPU e dois discos.

Quando um programa (processo) é submetido para execução no NeoLook, o sistema primeiro seleciona em qual computador ele será executado. Assume-se que um processo executa inteiramente em um *único computador*. A execução de um processo em um computador, quando iniciada, consiste basicamente na realização de operações na CPU (processamento), seguida da realização de operações em um dos discos (recuperação de dados) do computador e, por fim, da transmissão de dados pela rede (que é compartilhada por todos os computadores). No que segue, faremos referência a estas operações executadas em cada recurso (CPU, disco ou rede) por processamento/operação/execução no recurso.

A execução de operações de um processo p em um recurso r (CPU, disco, ou rede) ocorre da seguinte maneira:

- Se o recurso r está livre (não está alocado para nenhum outro processo, p assume o recurso r , executando durante um certo intervalo de tempo D_r^p . Durante este intervalo de tempo, o recurso r é considerado alocado para p , não estando livre para demais processos que tentem acessá-lo.
- Se o recurso r não está livre (atualmente está alocado para outro processo q), p entra em uma *fila de espera* associado a r , esperando pela sua vez para executar.
- Quando p finaliza sua execução em r , p libera o recurso r , e segue tentando acessar um dos outros recursos do sistema ou finaliza sua execução. Caso existam outros processos à espera por r (isto é, na fila de espera de r), um destes processos

é selecionado para executar em r . Seja este processo q . A partir deste instante, q irá executar no recurso r por um intervalo de tempo igual a D_r^q . O módulo que faz a seleção de qual processo na fila de espera de um recurso deve assumir sua execução chama-se **ESCALONADOR**.

Você pode assumir que cada processo p , ao ser direcionado para um computador do sistema para iniciar sua execução, primeiro busca realizar o processamento na CPU local. Este processamento gastará um tempo D_{CPU}^p . Após terminá-lo, o processo passa a recuperar dados de um dos dois discos do computador. Esta operação gasta um tempo D_{disco}^p . Por fim, os dados são transferidos para a rede, o que gasta um tempo D_{rede}^p .¹ Os intervalos de tempo D_{CPU}^p , D_{disco}^p e D_{rede}^p são chamados de *demandas* de p pelos recursos CPU, disco e rede, respectivamente. Após esta operação, a execução de p é considerada finalizada, e o processo deixa o sistema. Esta é uma simplificação significativa do sistema real, no qual operações de disco e de transferência pra rede são intercaladas com processamento na CPU. Porém, para fins da avaliação de desempenho pretendida, ela é razoável.

A seleção do computador no qual um dado processo deverá executar é feita de forma uniforme. Em outras palavras, qualquer um dos N computadores pode ser selecionado com igual probabilidade (no caso, com probabilidade $prob = 1/N$). *Você pode considerar que o tempo necessário para realizar esta seleção é desprezível para fins da avaliação de desempenho a ser realizada.*

De maneira similar, a seleção de qual dos dois discos do computador um processo deve acessar é feita de maneira uniforme: qualquer um dos dois podem ser acessados com igual probabilidade ($prob = 1/2 = 0.5$). Entretanto, um processo acessa apenas *um* dos dois discos. Em outra palavra, uma vez selecionado qual dos dois discos um processo p deve acessar, este executará por um intervalo de tempo igual a D_{disco}^p no disco selecionado (seguinte, ao final, para a operação na rede).

Os processos:

Cada processo que executa no NeoLook pode ter valores de demandas diferentes para cada recurso (ou seja, $D_{CPU}^p \neq D_{disco}^p \neq D_{rede}^p$). Além disto, processos diferentes podem ter demandas diferentes para um mesmo recurso ($D_{CPU}^p \neq D_{CPU}^q$). Entretanto a demanda por CPU D_{CPU}^p é a mesma, independente do computador no qual ele executará. De forma similar, a demanda por disco, dada por D_{disco}^p é a mesma, independente do disco que ele acessará.

Os processos são categorizados em três classes, a saber, HP (*high priority*), MP (*medium priority*) e LP (*low priority*) em função da sua prioridade. Esta prioridade poderá (ou não) ser utilizada para fins de escalonamento e gerenciamento das filas de espera (vide abaixo).

¹ Note que o recurso rede é compartilhado pelos processos executando em todos os N computadores do sistema (ou seja, existe um único recurso rede no NeoLook).

O escalonador:

Conforme mencionado, o escalonador toma decisões de gerenciamento de cada fila a fim de selecionar qual processo, dentre aqueles correntemente à espera por um dado recurso r , deve iniciar sua execução em r . A fila de espera de cada recurso deve ser mantida de forma a suportar estas decisões definidas a partir de uma **política de escalonamento**. As seguintes políticas devem ser consideradas:

- First Come, First Served (FCFS): esta é uma política tradicional de gerenciamento de filas, na qual cada novo processo é sempre inserido ao final da fila, no momento da sua chegada na mesma. Em outras palavras, ele terá que esperar que o processo correntemente em execução bem como todos os que foram inseridos na fila de espera anteriormente à sua chegada terminem sua execução em r para que possa iniciar sua operação.
- Shortest Job First (SJF): esta política garante que o próximo processo a executar é aquele que, dentre todos na fila de espera, tenha a menor demanda pelo recurso em questão. No caso de empate, o tempo de inserção na fila deve ser considerado como fator. Ou seja, sendo dois processos com mesma demanda, aquele à espera por mais tempo têm prioridade. Esta é uma política difícil de implementar na prática pois exige conhecimento prévio sobre por quanto tempo o processo executará no recurso (demanda), o que pode não ser conhecido e precisará ser estimado. Entretanto, pra fins de simulação, as demandas de cada processo em cada recurso serão conhecidas. Logo, a política pode ser avaliada.
- Priority (PRI): esta política considera as três classes HP, MP e LP dos processos. Processos da classe HP têm prioridade sobre processos da classe MP, que por sua vez têm prioridade sobre processos da classe LP. Neste caso, se um processo p , ao tentar acessar um recurso r , precisa esperar na fila (ou seja, um outro processo está atualmente executando neste recurso), p deverá ser inserido *na frente de todos os processos de menor prioridade que correntemente esperam na fila por r* . Em outras palavras, a fila de espera é mantida ordenada, em ordem decrescente de prioridade. Novamente, o tempo de espera na fila deve ser considerado como fator de desempate.

O simulador:

Você deverá construir um simulador do sistema NeoLook conforme especificação acima, a fim de realizar uma comparação do desempenho das três políticas de escalonamento discutidas (FCFS, SJF, PRI). O seu simulador deve capturar os detalhes do sistema alvo bem como dos processos discutidos acima. Em outras palavras, cada um dos N computadores deverá ser representado como 3 recursos (1 CPU, 2 discos), havendo um recurso para representar a rede compartilhada. Cada recurso deverá ser representado por uma fila de espera, gerenciada conforme política de escalonamento sendo avaliada. Portanto, o seu simulador deve implementar $3N+1$ filas, cada uma gerenciada (através da política de escalonamento) de maneira independente. Entretanto, você pode assumir que a mesma política de escalonamento será utilizada em todas as filas.

Processos são iniciados durante toda a simulação. Logo no início, um processo é atribuído a um computador (veja discussão abaixo). A partir daí, a execução deste processo será simulada através da execução do mesmo na CPU, seguida pela execução em um dos discos (selecionado conforme descrito abaixo) e por fim na rede, conforme discutido acima. Cada um destes passos deve ser simulado através da inserção/remoção do processo na respectiva fila. Note então que o evento associado ao início da execução de um processo em um computador encadeia uma sequência de eventos, a saber: ele é imediatamente inserido na fila da CPU; eventualmente, ao término da sua execução na CPU, ele é inserido na fila de um dos discos; ao término da execução no disco, ele é inserido na fila da rede, e, ao término da execução da rede, o processo é finalizado. Este último evento corresponde ao término da execução do processo.

Para realizar a simulação, você deverá manter uma lista de eventos, cada um especificando, dentre outras coisas, um tipo (*Qual o evento?*) e um instante de tempo (*Quando o evento ocorre?*). A lista de eventos deve então ser mantida ordenada pelo instante de ocorrência, de forma a garantir que eventos são tratados em ordem cronológica. A dinâmica geral do simulador consiste então em caminhar por esta lista de eventos, realizando o tratamento adequado de cada evento no instante de ocorrência, e adiantando um relógio lógico à medida em que o tempo (simulado) progride. Este relógio nada mais é do que um contador de *unidades de tempo*. Todas as demandas deverão ser especificadas na mesma unidade de tempo (p.ex: segundos).

Entrada da simulação:

A simulação deve receber como entrada dois parâmetros: (1) uma cadeia de caracteres especificando qual política de escalonamento deve ser utilizada (FCFS, SJF ou PRI), e (2) o nome de um arquivo contendo um *trace* (lista de registros) especificando os processos cujas execuções devem ser simuladas.

A política de escalonamento especificada deve ser utilizada em *todas* as filas simuladas.

Cada linha do *trace* especifica a execução de um processo, seguindo o formato:

$\langle instante \rangle \langle D_{cpu} \rangle \langle D_{disk} \rangle \langle D_{rede} \rangle \langle classe \rangle$

onde:

- $\langle instante \rangle$ especifica o momento em que o processo é iniciado. Este instante é um valor inteiro que representa o número de unidades de tempo decorridas desde um marco inicial (instante 0)
- $\langle D_{cpu} \rangle$ especifica a demanda, em unidades de tempo, de CPU do processo
- $\langle D_{disk} \rangle$ especifica a demanda, em unidades de tempo, de disco do processo
- $\langle D_{rede} \rangle$ especifica a demanda, em unidades de tempo, de rede do processo
- $\langle classe \rangle$ especifica a classe do processo. Esta classe é especificada por um valor inteiro, sendo 0 para classe HP, 1 para classe MP e 2 para classe LP.

Um exemplo de arquivo de entrada é:

```
0 10 20 3 1
10 2 15 5 1
12 5 50 30 2
.....
```

O exemplo especifica o início de um processo da classe MP no instante 0, com demandas na CPU, disco e rede iguais a 10, 20 e 3 (unidades de tempo), respectivamente. Um outro processo da classe MP é iniciado 10 unidades de tempo depois (instante 10). Este processo tem demandas na CPU, disco e rede iguais a 2, 15 e 5, respectivamente. Duas unidades de tempo depois (instante 12), um novo processo é iniciado. Este processo tem demandas por CPU, disco e rede iguais a 5, 50 e 30, respectivamente.

O arquivo pode ter um número arbitrário de linhas (isto é, processos).

O nome do arquivo de entrada deve ser passado como parâmetro na linha de comando. Em outras palavras, o seu programa deve ser executado como, por exemplo:

programa FCFS arquivo_de_entrada.txt

onde programa é o nome do seu executável e arquivo_de_entrada.txt é o nome do arquivo com o *trace* a ser simulado.

Para tanto você precisará utilizar primitivas para abertura, fechamento e leitura de arquivos, bem como para manipular parâmetros passados na linha de comando. Investigue o uso dos parâmetros `argc` e `argv` do `main`.

Saída da simulação:

O seu programa deve produzir e imprimir na tela estatísticas do desempenho do *trace* simulado. As estatísticas a serem computadas e impressas são:

- Tempo médio de execução dos processos no NeoLook: o tempo de execução de um processo inclui não somente os tempos de processamento em cada recurso mas também os tempos em que o processo ficou nas filas de espera associadas.
- Tempo médio de espera: o tempo de espera de um processo consiste na soma dos tempos de espera pela CPU, por disco e pela rede. Você deve contabilizar os tempos de espera de cada processo simulado e, ao final, computar o valor médio.
- Taxa de processamento: a taxa de processamento consiste na razão do número total de processos executados pelo intervalo de tempo decorrente desde o início do primeiro processo até o término do último. Logo, se o primeiro processo foi iniciado no instante t_1 e o n -ésimo (último) foi finalizado no instante t_2 , a taxa de processamento é calculada como $n / (t_2 - t_1)$.

Além disto, para simulação com a política PRI, você deve também calcular e imprimir as estatísticas de tempo médio de execução, tempo médio de espera e taxa de processamento separadas para cada uma das três classes de processo.

Seleção uniforme de computadores e discos:

A seleção do computador no qual um processo irá executar bem como de qual dos dois discos locais ele acessará é feita de forma probabilística seguindo uma distribuição uniforme, isto é, qualquer computador (disco) é igualmente provável de ser selecionado. Para fazer esta seleção, você precisará então fazer uso de um gerador de números pseudo-aleatórios.

Na linguagem C, você pode usar as funções *srand48(semente)* e *drand48()*, ambas definidas em *<stdlib.h>*. A função *srand48(semente)* é responsável por inicializar o gerador de números aleatórios e deve ser chamada no início do seu programa principal, antes de começar a simulação. O parâmetro *semente* é um valor inteiro qualquer. Passada uma mesma semente, você irá sempre gerar a mesma sequência de números. Logo, o parâmetro *semente* deve ser variado quando se quer variar os números gerados. Para fins da sua simulação, você pode usar o mesmo valor de semente sempre (qualquer valor inteiro).

A função *drand48()* retorna um valor de ponto flutuante não negativo com precisão dupla, uniformemente distribuído no intervalo [0, 1). Logo, para se gerar um número *inteiro* entre 0 e N-1, representando a seleção de um dos N computadores, basta chamar:

$$(\text{int})(\text{drand48}() * N)$$

Para fazer a seleção de um dos dois discos, basta então chamar $(\text{int})(\text{drand48}() * 2)$

Implementação e documentação:

A implementação deverá ser feita utilizando alocação dinâmica de memória (apontadores).

Você deverá fazer vários testes com o seu programa. Um trace de exemplo será fornecido, mas você deverá também gerar várias outras entradas de teste. Para cada trace de entrada, o aluno deverá simular cada uma das três políticas. O relatório deverá conter, além da descrição da estrutura geral do programa, estruturas de dados, algoritmos e ordem de complexidade de todas as funções, uma discussão do desempenho relativo das três políticas de escalonamento em termos das três métricas avaliadas (tempo médio de execução, tempo médio de espera, taxa de processamento). Para cada entrada testada: (1) qual política leva a um tempo de execução médio menor? (2) qual leva a uma taxa de processamento maior? (3) quando a política PRI é utilizada, qual é o desempenho médio relativo dos processos de cada classe (HP, MP, LP)?

Pontos extras:

As seguintes extensões do simulador e dos experimentos poderão ser realizadas valendo pontos extras:

- (1.5 ponto) Considere que cada processo pode esperar na fila de um recurso por um tempo máximo T_{\max} . Se T_{\max} unidades de tempo se passaram desde o instante em que um processo p foi inserido na fila de um recurso r sem que ele tenha começado seu processamento em r , assume-se que o processo deve ser finalizado, isto é, ele deve ser removido da fila de r e do sistema, sem passar pelos outros recursos que seriam visitados após r . Diz-se que p “morreu de inanição”. Estenda seu simulador para implementar e contabilizar o número de mortes por inanição em *cada fila do sistema*. O seu simulador deve imprimir como saída, além dos itens listados acima, também a *porcentagem* de mortes em *cada fila simulada*. Você deve fazer experimentos com diferentes *traces* de entrada (você deve gerá-los) e pelo menos 2 diferentes valores de T_{\max} . A discussão do desempenho relativo entre as políticas de escalonamento deve levar em consideração o número de mortes por inanição. Em outras palavras, qual o desempenho relativo das políticas considerando não somente o tempo médio de execução e a taxa de processamento, mas também o número de processos mortos por inanição?².
- (3 pontos) Uma outra política de escalonamento muito comumente utilizada é a chamada *Round Robin (RR)*. Na política RR, cada processo p , ao assumir um recurso r , executa por um certo intervalo de tempo Q chamado *quantum*. Terminado o quantum de execução, caso o processo p não tenha finalizado o processamento no recurso r (ou seja $D_r^p > Q$), ele é reinserido no *final da fila* de espera. Mais tarde, ao reassumir o recurso, p executará novamente por um intervalo Q ou até que termine a execução (se faltar menos de Q unidades de tempo para isto). Em outras palavras, as execuções dos processos em um recurso são intercaladas, sendo que cada processo recebe uma janela fixa Q^3 de tempo para executar. Se ele não conseguiu finalizar o processamento nesta janela de tempo, ele é reinserido na fila de execução, no final da mesma, devendo então esperar que todos os outros processos executem também por um quantum Q para que possa reassumir sua execução em r . Você deve implementar a política RR no seu simulador, considerando que o mesmo valor de Q é utilizado para todos os recursos. Você deve incluir a política RR na sua avaliação, comparando-a com FCFS, SJF e PRI considerando os diferentes *traces* de entrada. Faça experimentos com pelo menos 2 valores diferentes de Q .

Note que apenas extensões de simuladores que contemplam a especificação original serão avaliadas.

² Quanto maior o número de mortes, pior é a política considerando esta métrica.

³ O quantum Q é o mesmo para todos os processos executando em um recurso.